PRABUDDH : A PROTOTYPICAL KNOWLEDGE INTEGRATED EXPERT SYSTEM SHELL (VERSION I)

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

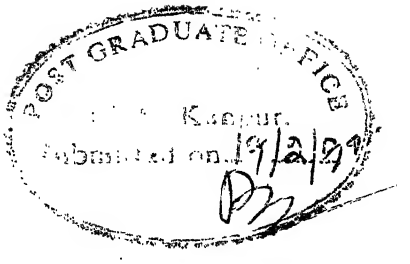
AKINEPALLI SREENATH

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
February, 1990

CSE-1990-M-SRE-PRA

.. To my beloved Parents.



CERTIFICATE

This is to certify that the present work `PRABUDDH : A Prototypical Knowledge Integrated Expert System Shell (Version I)' has been carried out by Mr. Akinepalli Sreenath under my supervision and it has not been submitted elsewhere.

(Dr. R.M.K. Sinha)
Professor,
Dept. of Comp. Sci. & Engg.,
I.I.T. Kanpur.

February, 1990

ACKNOWLEDGEMENTS

I am extremely grateful to **Dr. R.M.K. Sinha** for his expert guidance without which this work would have been a nightmare for me.

I am thankful to C.S.E. Lab Staff for their assistance and kind co-operation during this work.

I would like to express my sincere gratitude to Dr. Raj Kumar for his kind help in my thesis writing.

Last but never least, my note of special thanks to all my IITK friends and especially to the always ebullient clique ___ Sattu, Lambu, Pandu, Druid, Jaggu, Lambda, Nasy, Maamu, Jaggie, Thai, Shiv Kumar, Venu, Ravindra and Jafo ___ who made my stay at IITK not only enjoyable and exciting but also sometimes unendurable!

February, 1990 Akinepalli Sreenath

ABSTRACT

In the present work, an expert system shell that uses both framelike structures and rules to solve problems in knowledge intensive domains like medical diagnosis is implemented. The main objectives include building of an architecture which is capable of being tuned to a wide class of situations pertaining to diagnosis and to provide good user interface.

The knowledge representation scheme used is a combination of frames and rules into a single data structure called **prototype**. Many medical texts present diseases by discussing typical cases. It is relatively easy to express this kind of knowledge in terms of prototypes. Different types of knowledge can be separated in this formalism. The base idea is that the prototype represent typical case situations where as expressing the same knowledge in other formalisms like logic or production rules is not natural.

The inference mechanism supported by the shell can be viewed as "hypothesize and match" i.e. the basic problem solving paradigm involves seeing how well some (possibly idealized) representation of a hypothetical situation actually fits the facts. An explanation facility which provides an interactive view of the knowledge bases is present. Lastly, good user interface with facilities to create and update knowledge base is also available.

CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

One of the main reasons for the rapid growth in the rate of expert systems (ES) development is the support of rich set of powerful development tools. There are many different types of ES tools with a wide variety of levels and types of functionality and hardware support requirements. In general, ES tools can be thought of as comprising a spectrum. This spectrum moves from the lowest-level general-purpose programming languages to the highest-level large hybrid development environments.

The first major development in ES-specific tools was the introduction of the ES **shell**. The first example of a such a tool was Emycin [Shortliffe85], which grew out of the Mycin project. Emycin is a shell in the sense that it is an existing ES with the domain-specific knowledge removed. Emycin is rule-based with a backward-chaining inference strategy, and it is designed for diagnostic consultation tasks.

The concept of a shell gives way to the concept of a **narrow** ES tool. (In this case "narrow" implies specialized, and it should not be thought of in a pejorative sense.) Although these narrow tools were not developed by eviscerating an existing ES, they do have in common the characteristics that they are specialized in a particular form of processing (e.g. focus on one knowledge representation).

Tool Narrow/ Hybrid	Knowledge representation scheme	Highlights / Features with reference.
Emycin	Rule-based	Backward chaining inferencing strategy, diagnostic consultant Implemented in Lisp [Shortliffe85]
OPS5	Rule-based	Forward chaining inferencing used, Popular basis system, Rete-algorithm Implemented in Lisp [Brownston85]
Radial	Rule-based	Built-in handling of fuzzy logic. Implemented in C. [Michie84]
KL-ONE	Frame-based	Automatic inheritance Implemented in Lisp [Branchman85a]
Kandor	Frame-based	Small embedded system with simple user interface. Implemented in Lisp [Patel84]
Prolog	Logic-based	Logic-based general-purpose programming language [Clocksin84]
Mandala	Logic-based	Object-oriented programming [Furukawa84]
KEE	Large Hybrid Tool	Supports representation schemes like rules, frames & Logic. Both types of chaining & Inheritance are followed. Good User Interface and Built-in Explanation are also available. [Waterman86]
Loops	Large Hybrid Tool	Supports rules and frames. Both types of chaining& Inheritance are available. Other feature is Good User Interface. [Beach86]
Exsys	PC-Based Tool	Supports rules and backward chaining. Built-in explanation & Software Interface to Dbase & Lotus 123. [Karna85]

Table 1.1 Some ES tools with their Features

The most generally powerful ES tools are **large hybrid** tools that combine sophisticated development environments with multiple knowledge representations and multiple inference paradigms. A hybrid tool allows the use of a single tool to solve a large complex problem (Permaid [Rolston87]) when different parts of the problem require different types of support.

Table 1.1 covers the spectrum of ES tools along with their major features. Also VIDHI, which is a rule based backward chaining shell developed at I.I.T Kanpur can be given as an example.

1.1 The Pros and Cons of Major Knowledge Representation schemes used in ES shells

The three major formalisms of knowledge representation schemes which are used in most of the ES shells include rule-sets (Production system), generalized graphs (semantic nets, frames and objects) and predicate logic.

1.1.1 The pros and cons of predicate logic

Logic is one of the earliest forms of knowledge representation used. Logic is a kind of formal language, namely a calculus with syntactic rules of deduction. The most widely used formal logic system, called **first-order predicate** logic,consists of four major components: an alphabet, a formal language, a set of basic statements called axioms (expressed in the formal language),

and a set of inference rules. Each axiom describes one fragment of knowledge, and inference rules are applied to the axioms to derive new statements of truth. The most common rules of inference include modus ponens, universal specialization and substitution. There are several more powerful techniques for performing inference. Resolution, an example of such technique is very popular. Logic programming languages were developed which gives the knowledge engineer facilities for quantification and pattern matching.

You get backtracking and pattern matching for free in a logic programming language like PROLOG. However, you may not want the backtracking as standard, and the pattern matching provided may not meet your precise requirements. For many applications, though, these facilities are welcome and useful.

It is not altogether trivial to structure rules and data according to your needs, although it can undoubtedly be done by an experienced programmer. The typical resolution refutation system deals with a set of propositions and uses only minimal indexing. Large non-deterministic programs really require some extra control features to be built into a special-purpose interpreter.

1.1.2 The Pros and Cons of production rules

Production rules are a formalism which in the expert systems literature, are sometimes called `condition-action rules' or `situation-action rules'. Their principal use is in the encoding of empirical associations between patterns of data presented to

the system and actions that the system should perform as a consequence. A production system consists of a rule set (sometimes called `production memory'), a rule interpreter that decides how and when to apply which rules, and a `working memory' that can hold data, goals or intermediate results.

The production systems typically do forward chaining i.e., matching data in working memory against the left hand sides of rules and then executing the right hand sides of the rules. However a more goal-driven effect, i.e. backward chaining effect can be achieved by having task statements in working memory and making the first condition of each rule a test for the current task.

Most production rule interpreters don't support back-tracking since modifications to data structures are destructive, making it difficult to return to an earlier state of the computation. Control mechanisms are global and context-free. This has simplicity on its side, but there are times when it is both inefficient and restrictive.

1.1.3 Pros and cons of the structured objects

The structured objects include the semantic nets, frames and objects.

`Semantic nets' are a kind of network commonly used to structure more general kinds of information. These are constellations of nodes and links in which the nodes stand for concepts and the links stand for relationships between them. The

term `associative nets' is most commonly used in synonym with semantic nets.

Frames, by contrast are a way of grouping information in terms of a record of `slots' and `fillers'. This record can be thought as a complex node in the network, with a special slot filled by the name of the object that the node stands for, and the other slots being filled with the values of various common attributes associated with such an object. Frames are described as `data structures for representing stereotyped situations'. These are also called as `prototypical objects 'or `prototypes'. These are the most important of all the structured objects.

The building blocks of an object oriented representation were called `conceptual objects' and were similar to the frames in that they stood for prototypes and their associated properties. This representation is still in the research stages.

Frames are good implementation devices for exploiting inherent structure in rules and data. They also lend themselves to model-fitting approaches, e.g. looking for a best match between data and some hypothesis. Context dependent interpretation and local flow of control can be very useful for certain applications.

Like production systems, frame systems don't usually support backtracking for the same reason. Frames offer mechanisms for dealing with things like exceptions and defaults which are not easy to handle in standard logic. However they are often criticized precisely because of this, as the cancellation and alteration of inherited properties make it difficult to properly

define any thing in terms of anything else. Many people are unhappy with frame- and object-based systems because they seem to depart from logic and because their flexibility in matters of context and control can make their behaviour both hard to predict and difficult to understand.

1.2 Motivation

Much of artificial intelligence research has focused on determining the appropriate knowledge representations to use in order to achieve high performance from knowledge-based systems. It has been observed that using more than one knowledge representation structure has two main advantages.

1. It is not necessary to force all that is known about the domain into a single codification scheme.

2. Different types of knowledge can be represented in the formalism.

The chosen scheme or schemes should be expressive enough to represent different kinds of knowledge and render them explicit so that they can be manipulated by the program.

Conflict resolution which is provided in the production system, suffers from two main troubles from the present point of view.

1. The mechanism is coded into the interpreter, and therefore difficult to modify, although options are often supplied (as in OPS5).

2. The strategic knowledge implicit in the conflict

resolution algorithm is nowhere represented explicitly, and therefore cannot be reasoned about by other parts of the program.

The very modularity and uniformity of production rules, which have been cited as positive features, have their negative aspect. Most rule sets contain implicit groupings, either using various kinds of indexing concealed in the interpreter (e.g. the ORGRULES and PATRULES of MYCIN), or having conditions and actions that manipulated goal tokens in working memory (e.g. the context symbols of R1). Many of the problems cited above can be traced to the failure to differentiate between the different kinds of knowledge, which may need to be represented and applied in different ways.

In the present work, an expert system shell that uses both framelike structures and rules to solve problems in knowledge-intensive domains like medical diagnosis is being implemented. The main objectives include building of an architecture which is capable of being tuned to encapsulate knowledge about a very wide class of situations and to provide good user interface facilities.

1.3 CURRENT IMPLEMENTATION

In the expert system shell implemented, the knowledge representation scheme used is a combination of frames and rules. This data structure is called a PROTOTYPE. A Prototype represents a typical case situation and it is relatively easy to express the medical expertise in terms of prototypes. The basic idea is that frame like structures provide an explicit representation of the

context in which production rules do their reasoning. This allows one to separate strategic knowledge about what can be inferred from what set of facts. In theory, this allows inferential knowledge to be put to different contexts, with gains in both economy & coherence for the knowledge representation.

Thus production rules can be conceived as simply one kind of value for a particular kind of slot in a frame. Associating rules with slots in frames provides one mechanism for organizing rules into what appears to be natural groupings. The other slots in a given frame provide the explicit context in which its rules are applied.

The prototypes are arranged in a network in which the links specify the relationships between the prototypes. They contain both object & meta-level knowledge. Each prototype has slots for a number of components, which point to subframes of knowledge at the object level. Each component is again a frame-like structure in its own right, with its own internal structure. Also prototypes contain control slots, which enable the domain-expert to incorporate information about how to reason with this knowledge structures.

The inferencing mechanism supported by the shell can be structured as `hypothesis & match', .i.e. the basic problem solving paradigm involves seeing how well some (possibly idealized) representation of a hypothetical situation actually fits the facts. This is done by attempting to reinstantiate prototypes in order to determine the goodness of the match, and it

may involve obtaining more information from the user. A run of the consultation of the system developed on the shell appears as an interpreter executing an agenda of tasks. The other salient features of inference mechanism include backtracking which enables the user to have alternate solutions and an explanation facility.

Lastly, good User interface is available at the user's disposal which enables him to construct & edit the knowledge bases easily. The editing facility includes various functions that guide the user through the process of expanding or modifying the knowledge base.

1.4 ORGANIZATION OF THESIS

In chapter 2, a review of the related work i.e. salient features of the expert system architectures which were built using similar knowledge representation scheme & inference mechanisms (multiparadigm architectures) is presented.

In chapter 3, an overview of the implemented expert system shell architecture is presented.

Chapter 4 discusses the implementation details of the various modules present in the shell architecture.

In chapter 5, conclusions and the scope for future work are presented.

CHAPTER 2

MULTIPARADIGM ARCHITECTURES: A REVIEW

In this chapter, the salient features of some of the multiparadigm expert systems architectures which influenced the expert system shell developed are presented.

2.1 CENTAUR

CENTAUR [Aikins83] is a consultation system that produces an interpretation of data & a diagnosis based on a set of test results. The inputs to the system are the pulmonary function test results and a set of patient data including the patient's name, age etc. and a referential diagnosis. The output consists of both a set of interpretation statements that serve to explain or comment on the pulmonary function test results and a final diagnosis of pulmonary disease in the patient.

2.1.1 Knowledge representation in the CENTAUR

Knowledge is represented in CENTAUR by both rules and prototypes. Each prototype contain two kinds of information, domain-specific components that express the substantive characteristics of each prototype, and domain-independent slots that specify information used in running the system. Each component may, in turn, have slots of information associated with it, including a RULES slot that links the component to rules that determine values of the component.

Most of CENTAUR's prototypes represent the characteristic features of some pulmonary disease. There are components for many of the pulmonary function tests in each prototype that are useful in characterizing a patient with that disease.

The other important fields in a component include range of plausible values, possible error values, default value etc. In addition to domain-specific components, each prototype contains slots for general information associated with it. This includes book keeping information and English phrases used in communicating with the user.

The CENTAUR knowledge base also includes rules, which are grouped into different sets according to their functions. Some of them make conclusions about values of the components. Many of the rules are classified as **patient rules**, rules dealing with the patient. Those rules whose actions make summary statements about the results of the pulmonary function tests are classified as **summary rules**, rules that suggest general disease categories in their actions are classified as **triggering rules** and lastly those rules that are used in a second stage of processing, after the system has formulated lists of confirmed & disproved prototypes are called **refinement rules**.

In CENTAUR, each piece of case-specific data that has been acquired either initially from the patient's pulmonary function test results or later during the interpretation process is called a fact. Each fact has six fields of information associated with it which include its name, value, certainty factor, source of

generation (i.e. from user, default value etc.), type of fact (plausible value, error value, or surprise value) and the prototype name which accounts for that fact.

2.1.2 Control Structure for CENTAUR

CENTAUR uses a hypothesis-directed approach to problem solving. The goal of the system is to confirm that one or more of the prototypes in the prototype network match the data in an actual case. The final set of confirmed prototypes is the system's solution for classifying the data in that case. This set represents the diagnosis of pulmonary disease in the patient.

The system begins by accepting the test & patient data. Data entered in the system suggest or "trigger" one or more of the prototypes. The triggered prototypes are placed on a hypothesis list and are ordered according to how closely they match the data. The prototype that matches the data most closely is selected to be the current prototype, the system's current best hypothesis about how to classify the data in the case. The process repeats & the system moves through the prototype network confirming or disproving disease prototypes. The attempt to match data and prototypes continues until the system has concluded that it cannot account for any more of the data.

The control information used by CENTAUR is contained either in slots that are associated with the individual prototypes or in a simple interpreter. Some control strategies are specific to an individual prototype & need to be associated with it, while more

general system control information is more effectively expressed in the interpreter. Basically, the interpreter attempts to match one or more of the prototypes with the data in an actual case i.e. tries to instantiate one or more prototypes. The exact method to be used in instantiating the prototype depends on the individual prototype & is expressed in one of the prototype control slots.

Four of the slots associated with a prototype contain clauses that are executed by the system at specific times to control the consultation. Each clause expresses some action to be taken by the system at different stages: (a) in order to instantiate the prototype (CONTROL SLOT), (b) upon confirmation of the prototype (IF-CONFIRMED slot), (c) in the event that prototype is disproved (IF-DISPROVED slot), and (d) in the clean up phase after the system processing has been completed (ACTION slot).

2.1.3 Explanation & knowledge Representation in CENTAUR

CENTAUR asks questions of the user if it fails to deduce some piece of information it needs from its rules, or if it needs a value for a parameter that is explicitly labeled `ask-first'. Questions asked of the system by the user include HOW & WHY, but with a stage-dependent interpretation. CENTAUR always displays the current prototype, so that the context of the question & answer are clear.

CENTAUR has a lisp function called PROTOTYPEMAKER for addi prototypes to the knowledge base. Modifications to exist

prototypes is achieved simply by using the INTERLISP record editor.

2.1.4 Criticism

In CENTAUR, though an attempt was made to correct representational deficiencies by using prototypes, a more complex control mechanism was also introduced. It made use of triggering rules for suggesting and ordering system goals, and included an additional attention-focusing mechanism by using frames as an index into the set of relevant rules. It should be pointed out that CENTAUR contains fewer than 30 prototypes, and it is not altogether clear what would happen as the number of prototypes increased. The potential for complexity in terms of both relationships between prototypes and competition between them for a place on the hypothesis list could conceivably lead to the kind of scheduling and focusing problems that arose with earlier versions of INTERNIST.

2.2 PERMAID

Permaid [Rolston87] is a multiparadigm ES that is used for diagnosis and maintenance of approximately 10,000 large fixed-head disk subsystems. It is intended to demonstrate how the various concepts described in theory are applied in real systems (including the compromises that must be made to fit the theory in reality). The use of rule-based and network/frame-based reasoning is emphasized in this discussion. Permaid performs three primary

functions:

* Troubleshooting of observed problems.

* Predictive maintenance

* Media and File recovery.

It also provides training as an important secondary function.

The heart of the system is a "fault-resolution kernel," which is used to identify faults on the basis of symptom observations and specific test results. The knowledge representation for this kernel is based on the use of frames and networks, and inference for this section is based on abduction [Pople82].

The kernel can be entered in several different ways. The kernel is entered in an "identity suspected fault" mode when predictive maintenance processing suggests that there is probably a fault in the subsystem. The kernel can also be entered in a "find secondary fault" mode, which is used to determine whether any permanent secondary ("ripple effect") faults have been caused by the occurrence of a "primary" fault. The predictive maintenance section uses rules and forward-directed reasoning based on pattern recognition.

2.2.1 Knowledge Representation in Permaid

The kernel knowledge base is a **semantic (or associate) network** that is composed of two segments: the **classification segment** that breaks down general error classes (e.g., "error message") into specific symptoms and the **cause-effect segment**

which is used to collect the initial problem symptoms. The remainder of the kernel knowledge base is implemented by using cause-effect analysis to form an associative network made up of cause of relationships. The network takes the form of a **directed acyclic graph** (DAG) that represents the cause-effect relationships between the nodes on the graph.

2.2.2 Fault Resolution Kernel and Kernel Inference

The fault resolution kernel is based on the concept of **abductive inference**. The fault resolution kernel applies this concept in the form of cause-effect reasoning in which specific causes are hypothesized to explain observed events. A characteristic of most diagnostic domains is the occurrence of "cascading errors" [Pau86]: The effect of the primary fault is to cause some intermediate problem which in turn causes another intermediate problem until visible symptoms are eventually produced.

The process of inference begins with the collection of initial symptoms. The initial symptoms are then evaluated to select one primary symptom that will be the starting node for the search process. The selection process operates by first identifying the most probable cause of all the observed symptoms. The symptom that is selected as the start symptom is any symptom that is an effect of the most probable cause. The remaining symptoms are placed on the symptom list ordered by their Collapsed Decision Factors (CDFs). Ties that occur anywhere in this process

are broken by arbitrary selection.

Once the primary symptom has been identified, the process is based on the systematic selection and test of possible causes for a given effect i.e. a depth-first search of the cause-effect graph. In the selection of the best node to analyze at any point, the causes that connect to multiple symptoms are preferred. Once the cause of a given effect is identified, the process moves to the new cause and is recursively restarted with the newly identified cause being treated as an effect.

Another important feature of Permaid is the provision for an "unknown" response. An unknown response results from the fact that the user is unable to gather test results for some reason. This option allows Permaid to continue processing in the face of uncertain or incomplete information. This feature allows the users to provide honest answers to the questions.

2.2.3 Explanation, User Interface and Support Tools

Permaid supplies explanation information that is tailored for four different types of user: expert, knowledge engineer, specialist user, and field service engineer (the typical end user). Explanation for the knowledge engineer makes use of the high-resolution, bit-mapped, window-oriented graphics and mouse on the work-station and is intended to assist in debugging the system. This explanation system provides an interactive view of the knowledge bases during execution.

The development and maintenance of Permaid is supported by a

powerful graphic knowledge base editor and automated correctness and validation software. The graphics editor is used to construct and review the knowledge bases. The verification system is used to perform a variety of consistency and completeness checks as well as utility functions.

2.3 Other Related Works

In WHEEZE [Shoreliffe85] an attempt to develop an expert system which provides a uniform declarative representation for the domain knowledge and to permit additional control flexibility by the use of frame representation and agenda-based control scheme respectively is explored. [Jackson86] provides an excellent description of the basic issues in expert systems research.

CHAPTER 3

SYSTEM OVERVIEW

In this chapter an overview of the expert system shell architecture developed is presented. First, we examine the building blocks of a typical expert system shell architecture. This is followed by a module by module description of the proposed system.

3.1 Typical Shell Architecture

Building knowledge-based, or expert systems from scratch can be very time-consuming because assembling the knowledge-base and constructing a working program for a particular domain is a difficult, continuous task that can often get extended over several years (like the MYCIN project). This suggests the need for development of general tools to aid in the construction of knowledge based systems. An expert system shell can be described as an effective domain-independent framework which aids in the construction of expert systems. A typical expert system shell architecture is shown in Figure 3.1.

From the Figure 3.1, we can identify the important modules that are present in a typical shell. They are (a) **Knowledge Base Construction Aids** which provides the basic features necessary for the construction of a knowledge base. These features include the tools for implementing knowledge representation scheme (e.g. rule-based etc.) and editing operations like adding, deleting, and

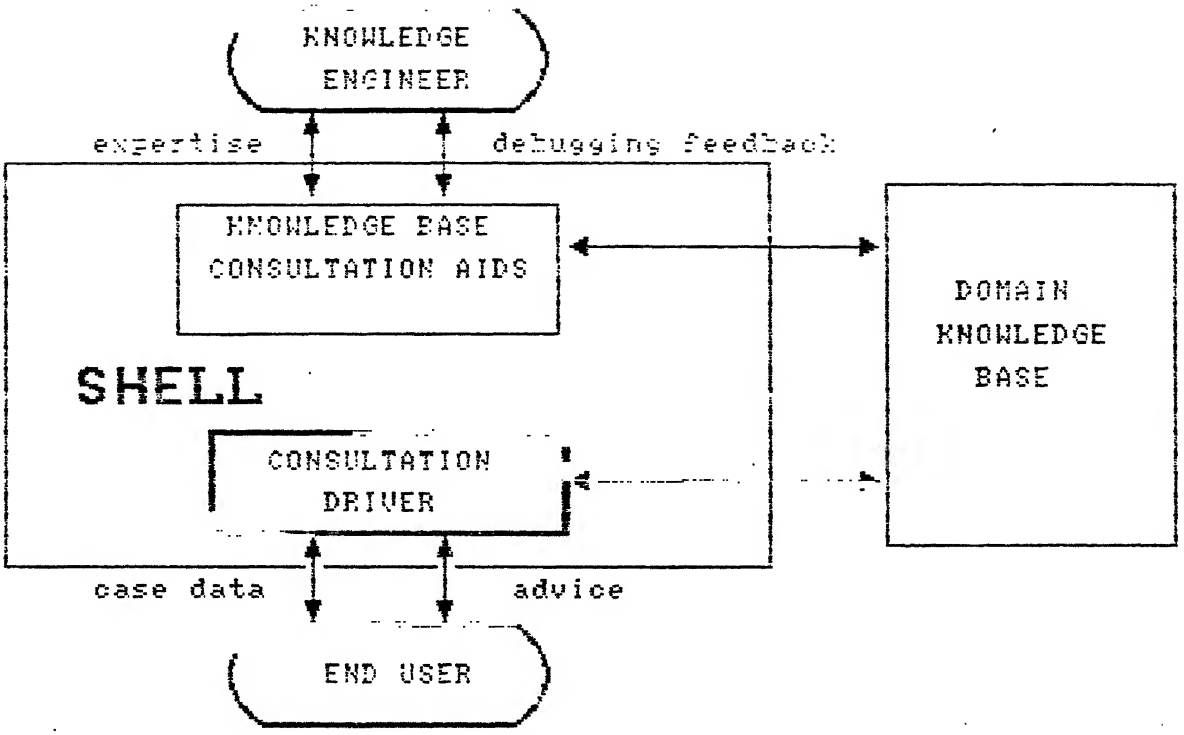


Figure 3.1 A Typical Shell Architecture.

modifying. (b) **Consultation Driver** which includes the inference engine and an explanation facility if available.

The knowledge engineer interacts with the shell to produce a knowledge base for the domain. The shell then interprets this knowledge base to provide advice to the end user. Thus the combination of the shell and a specific knowledge base of domain expertise is a new expert system.

The knowledge base and the inference engine form the heart of the new expert system. In general, knowledge is stored in a specific representation scheme and the schemes used for storing the information vary greatly. The design of this knowledge representation scheme has a good impact on the design of the inference engine, the knowledge updating process, the explanation process, and the overall efficiency of the system.

3.1.1 Selection of Appropriate Knowledge Representation Scheme

When the shell architecture is to be tuned to a application like medical diagnosis, the necessity of selecting an appropriate knowledge representation can be understood. Many medical texts present diseases by discussing typical cases. Trying to represent this kind of information using either rules or logic poses difficult problems. To handle exceptions using rules alone, one would have to cover all special cases in the premises: a clumsy and unacceptable solution if applied on a grand scale. To handle exceptions using logic, one engages in non-monotonic reasoning, i.e. one must be prepared to retract `old' propositions as well as

assert `new' ones. This complicates matters considerably, and neither the formal nor the computational ramifications of having to resolve such conflicts is well understood.

The notion of frame is very general. They are capable of representing information about `typical' objects and situations. One of the arguments for the use of frames is that they allow one to augment domain knowledge with knowledge about the context in which such knowledge is applied. For example, experts have knowledge about the inherent reliability of data that has been gathered in different ways. They also have expectations about patterns in the data, and can detect obvious inconsistencies.

Another argument is for using more than one knowledge representation structure. Here, there is no concern for a particular kind of knowledge structure. But the chosen scheme or schemes should be expressive enough to represent different kinds of knowledge and render them explicit so that they can be manipulated by the program.

3.1.2 Selection of Inference Mechanism and Others

As mentioned earlier, since the design of the inference engine is dependent on the knowledge representation schemes, selection of an appropriate inferencing mechanism cannot be carried out as an independent process. A brief description of the general inferencing mechanisms usually followed with different representation schemes is presented below.

If the representation scheme is logic, the most common

inference rule that is employed is the `Resolution' principle [Robinson65]. Resolution is potentially powerful and can be regarded as the generalization of more familiar rules, such as `modus ponens', `modus tollens' and chaining.

If the representation scheme employed is production rules, then the inferencing mechanisms followed include the forward and backward chaining. Forward chaining involves matching data in working memory against the left-hand sides of rules and then executing the right-hand sides while the backward chaining involves a goal driven effect.

If a frame based representation is selected, then the usual inferencing mechanisms which are followed are either a hypothesis-directed approach or an agenda-based control scheme.

Lastly, the user interface could be such that it enables the knowledge engineer to construct and edit the knowledge bases easily. The explanation facility could incorporate features like providing an interactive view of knowledge bases during execution, answering questions posed by the user within a broader context etc.

3.2 The Proposed Shell Architecture

The various modules of the proposed expert system shell are shown in Figure 3.2. The modules which are enclosed in the dotted box form the architecture of the shell. The knowledge engineer after interacting with the expert extracts the domain knowledge. The domain knowledge base is the result of expressing the domain

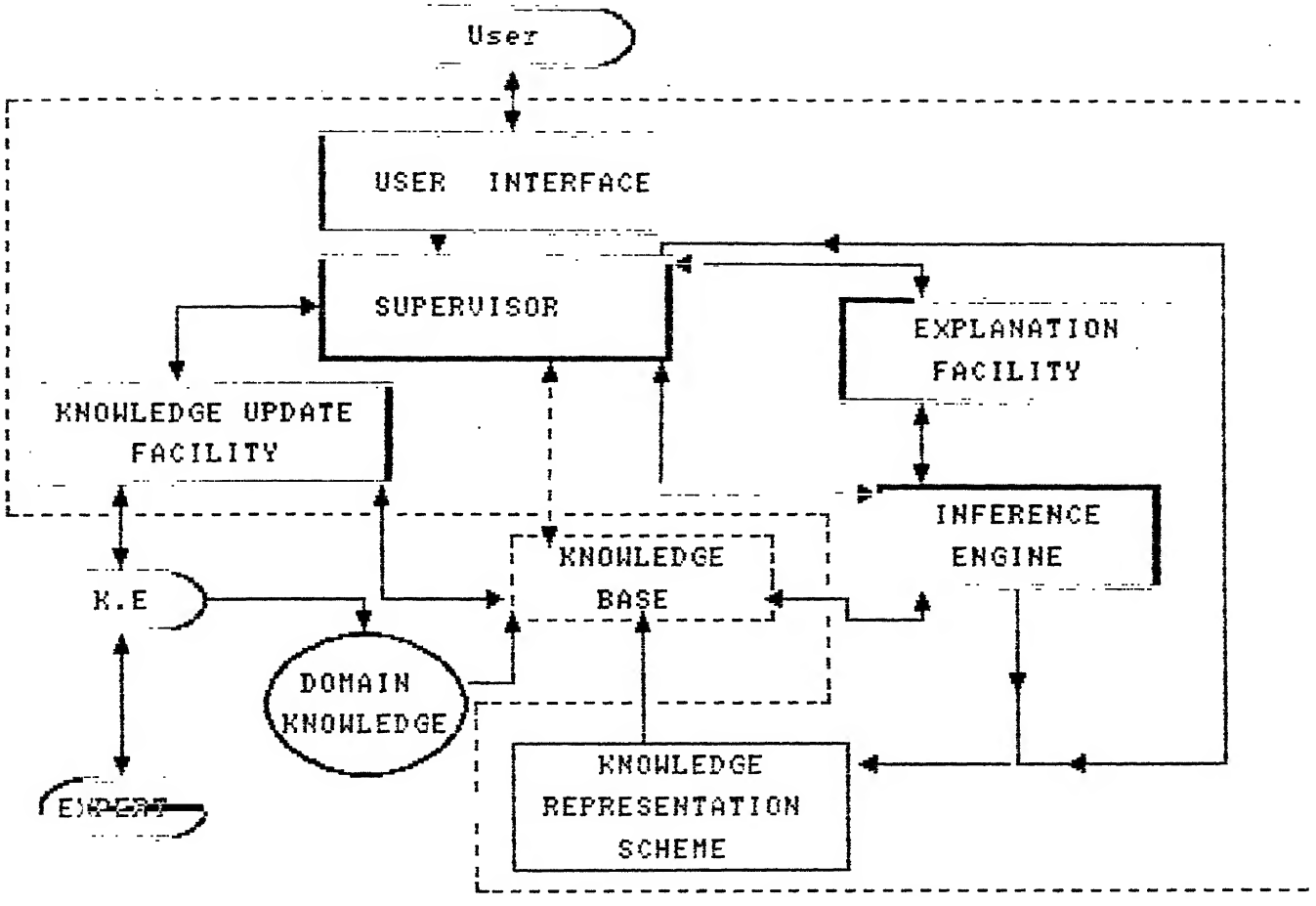


Figure 3.2 THE PROPOSED SHELL ARCHITECTURE.

knowledge in the knowledge representation scheme available. Then the combination of domain knowledge base and the shell results in a new expert system for that particular domain.

The important modules that constitute the shell architecture are:

Supervisor,

Knowledge Representation Scheme,

Inference Engine,

Explanation Facility,

Knowledge Update Facility &

User Interface.

In the following sub-sections, a brief description about each module is given to put the overall system in a proper perspective.

3.2.1 Supervisor

This module controls and co-ordinates the activities of various modules in the shell. The control is not centralized but distributed in nature. Each module gains the control when it is invoked and passes it to the next when it completes its function. The co-ordination activities include context switching from one inferencing strategy to another, linking the explanation facility with the inference process etc.

3.2.2 Knowledge Representation Scheme

The knowledge representation scheme used is a combination of frames and rules into a single data structure called prototype.

The whole idea is that they should be capable of being used to capture knowledge about a wide class of situations. Another advantage of prototypes from the programming point of view is that they provide a way of controlling interactions between modular units of knowledge. The prototypes are arranged in a network in which the links specify the relationships between them.

Each prototype has slots for a number of components, which point to subframes of knowledge at the object level. Each component is again a frame-like structure in its own right, with its own internal structure. Each component has fields like its associated name, important measure, default value etc. In addition, a component frame often contains two special slots called "value-rules" and "trigger-rules", which holds a set of production rules for inferring a value for that component and for triggering other prototypes respectively. Thus, in addition to having production rules embedded in prototypes, it is true to say that there are also prototypes embedded in other prototypes. This introduces another dimension of organization in the arrangement of prototypes in addition to the hierarchial one.

Also, prototypes contain control slots, which enable the knowledge engineer to incorporate information about how to reason with these knowledge structures. These slots hold clauses for:

* instantiating a prototype, by specifying a set of components for which values should be determined;

* confirming a prototype, by specifying the confirmation criterion;

* printing statements that summarize the final conclusion.

Each control slot can be considered as the consequent part of a rule whose condition is that the situation must match that described by the prototype.

An example prototype named GENERAL SYMPTOMS, a portion of which is shown in Figure 3.3, would enable you to understand easily, the prototype structure described above.

PROTOTYPE	GENERAL SYMPTOMS
GENERAL INFORMATION	
--English Phrase	"This frame collects the General Symptoms needed"
--Link prototypes	(Heart-Diseases Infectious-Diseases Normal)
COMPONENTS	BODY TEMPERATURE
range of values	Range: (96.0 110.0)
Default value	Default: 98.6
Value_rules	Val-rules: BTEMP_VR1, BTEMP_VR2
Trigger rules	Trig-rues: BTEMP_TR1, BTEMP_TR2
Plausible values	
.	BLOOD PRESSURE
.	Plausible-vals: (120/80 110/60 ..)
.	Default: 120/80
	Val-rules: BP1, BP2, BP3
CONTROL INFORMATION	
--Order of filling	(BLOOD PRESSURE, BODY TEMPERATURE, ..)
ACTION INFORMATION	"Some of the GENERAL SYMPTOMS were confirmed."

Figure 3.3 A Sample prototype `GENERAL SYMPTOMS'

3.2.3 Inference Engine

The approach being used in the shell inferencing mechanism can be characterized as "hypothesize and match", i.e. the basic problem solving paradigm involves seeing how well some (possibly idealized) representation of a hypothetical situation actually fits the facts. this is done by attempting to instantiate prototypes in order to determine the goodness of the match, and it may involve obtaining more information from the user.

A run of the consultation of the system developed on this shell consists of an interpreter executing an agenda of tasks. Each task is an action, specified by a call to a LISP function. New tasks are added to the agenda by prototype control slots and by other tasks (of whom they are the subtasks).

A primary use for the agenda is to provide an explanation of why the system behaved as it did in the course of the consultation. Consequently, each task entry contains information about both the source of the task and the reason that the task was scheduled. The source of the task will be a prototype or another task, since the tasks are added to the agenda either by prototype control slots or in the course of executing tasks already on the agenda. The reasons are generated from the name of the prototype and the name of the control slot responsible for setting up the task. However, there are some general tasks which are not specific to any one prototype, such as `order the hypothesis list', and these have text strings associated with them which give reasons for their use.

During program execution, prototype is always in one of three states:

* inactive, i.e. not being considered as a hypothesis;
* potentially relevant, i.e. suggested by data values;
* active, i.e. selected from the above and placed on the hypothesis list.

The hypothesis list is simply a list of prototypes paired with their degrees of suggestivity (explained in detail in the next chapter), ordered in deceasing order of their suggestivity. In other words, hypothesis list contains the set of triggered prototypes i.e. prototypes that are potentially relevant. Other lists keep track of confirmed and disconfirmed prototypes.

The key stages in the inferencing are:

* filling the initial data;
* triggering the prototypes using trigger rules;
* scoring the prototypes and selecting one to be `current';
* using known facts to fill the current prototype;
* confirming the current prototype;
* refining the hypothesis list accordingly;
* summarizing and printing the results.

Thus the inferencing proceeds in stages. A facility to have alternate solutions is also present.

The inferencing process described above is made further clear by considering the example prototype network shown in the Figure 3.4.

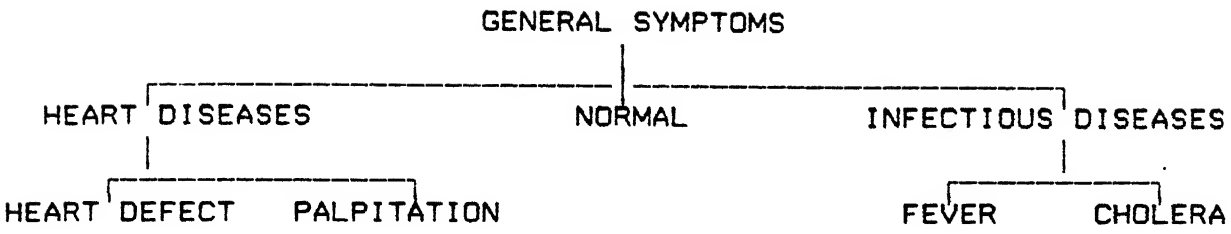


Figure 3.4 A Sample PROTOTYPE Network

In the prototype network shown in Figure 3.4, at the top of the hierarchy is the GENERAL SYMPTOMS prototype, which effectively controls the way in which the inference proceeds through various stages, such as initial data collection and the triggering of other prototypes. Then there is a layer of prototypes which represent different disease categories, such as HEART DISEASES, NORMAL, and INFECTIOUS DISEASES. Finally, diseases are specialized according to their subtype. Thus HEART DISEASES can be categorized as HEART DEFECT or PALPITATION, while INFECTIOUS DISEASES as FEVER or CHOLERA.

To begin the inference process, the GENERAL SYMPTOMS prototype is selected as the current prototype which elicits the initial data required. The components of this prototype may trigger other prototypes as they are being filled up. The triggered prototypes are placed on the hypothesis list ordered according to their degrees of suggestivity. Assuming that the prototypes NORMAL and INFECTIOUS DISEASES are triggered with degrees of suggestivity 45 and 70 respectively, the next prototype that would be made as the current prototype is INFECTIOUS

DISEASES. The inferencing proceeds to confirm the INFECTIOUS DISEASES prototype and its component values may trigger the prototypes FEVER or CHOLERA. Eventually, the system may confirm one of the FEVER or CHOLERA prototypes assuming that the prototype INFECTIOUS DISEASES is confirmed.

Since there are no more link prototypes at this stage, the system concludes that it has reached a solution. Now the summary statements associated with the confirmed prototypes are printed as conclusions and the user is prompted for another solution. If another solution is asked then the system continues with the remaining list of triggered prototypes until the list of triggered prototypes is exhausted or another solution is obtained, else the process ends.

3.2.4 Explanation Facility

In order to understand the inferencing process, in terms of both the direction that it takes from the user and the results that it returns to the user, a user must be able to understand:

* the question being asked;
* the reasons for asking them;
* the justification for intermediate conclusions.

The system asks user if it needs a value for a parameter that is explicitly labeled `askuser' during the inferencing process. Questions asked of the system by the user include HOW and WHY. Thus, a WHY question asked in the context of a particular prototype during the inferencing stage will be interpreted as "Why

are you asking for this information?". In the review stage, however, such a question would be interpreted as "Why are you considering this prototype?". The HOW question can only be asked in the context of a particular prototype. It will be interpreted as "How did you deduce the value of a particular component?".

For example, consider the prototype network shown in the Fig 3.4. The assumption is that the GENERAL SYMPTOMS prototype has been confirmed and it triggered the NORMAL prototype. In the process of filling up the components of the NORMAL prototype, if the user is asked for some information, and if the WHY question is posed by the user, it is interpreted as "Why are you asking for this information?". It is answered by saying that the system is trying to confirm the current prototype NORMAL. The list of components of NORMAL which are filled till then are also displayed. If the WHY question is posed again, then it is interpreted as "Why are you considering the prototype NORMAL?". The second WHY is answered by displaying the trigger rule of GENERAL SYMPTOMS which actually triggered the NORMAL prototype.

If the HOW question is posed by the user in the context of the NORMAL prototype, then the name of the component to be explained is enquired and the value-rule of the component which filled its value is displayed as the explanation.

3.2.5 Knowledge Update Facility and the User Interface

The Knowledge Update Facility makes use of the high-resolution, bit-mapped, window-oriented graphics and mouse on the

workstation, which enables the knowledge engineer to construct and edit the knowledge bases easily. The editing facility includes various functions that guide the knowledge engineer through the process of expanding or modifying the knowledge base.

A good user interface which provides an interactive view of the knowledge bases during execution is at the end user's disposal. A graphical display of the knowledge base is also available. As the inference process proceeds, nodes in the graph are highlighted (i.e. the current prototype is boxed). The explanation is provided in a separate window. Snapshots of the graphical display and the explanation window are included in the User's Manual for reference.

CHAPTER 4

IMPLEMENTATION DETAILS

In this chapter, the implementation details of the various system modules of the expert system shell are presented. The expert system shell is developed on NEXUS 3000 DOMAIN Workstation in LISP making use of the graphic features available on the workstation.

Before discussing the implementation of various modules, a brief description of the data structures used is presented.

4.1 Data Structures

The data structures used in the implementation of the expert system shell are shown in Figure 4.1.

4.1.1 Prototype

The main data structure is a prototype. Prototype structure is defined as a frame structure. Briefly the important fields in a prototype structure are as follows:

* **Eng_phrase**, which contains statement that describes about the prototype. This is used in communicating with the user.

* **Link_frames**, which contains a list of the prototypes at the next level that are related to this prototype.

* **Order_of_filling**, contains information about the order in which the components of the prototype are to be filled. If no specific order is provided by the knowledge engineer, a default

strategy in which the components are ordered according to their importance measure is followed.

* **Threshold_measure**, a parameter used in the confirmation of the prototype. It is provided by the domain-expert and is described in detail in the inference process. Its value must be in the range of 0 to 1000.

* **Action_info**, which contains summarizing or data interpretation statements which are printed in the conclusions if this prototype is among the confirmed prototypes list.

* **Components** contains the list of components defined for this prototype. Each component has its own structure which is explained in the next section.

* **Cert_measure**, the certainty with which the prototype can be confirmed when all its components are filled and confirmed. This measure is also provided by the domain-expert. This must be in the range of 0 to 100. This is absent for the root_frame.

* **Deg_of_belief**, the certainty measure observed in a particular case. This is a function of importance measures of the components, percent-of-belief of the components and certainty measure of the particular prototype. This must be in the range of 0 to 100. This is not considered for the root_frame.

```
;-----------------------------------------------------------------
   (defstruct frame  ;structure of the prototype is being defined.
           eng_phrase  ;English phrase which describes about the
                       ;prototype.
           name_of _frame  ;Associated name of the prototype.
           link_frames  ;contains the list of related prototypes.
           order_of_filling  ;contains information about the order
                             ;in which the components of this
                             ;prototype are to be filled.
           threshold_measure  ;parameter used in the confirmation
                              ;of the prototype.
           action_info  ;contains summarizing statements.
           no_of_comps  ;indicates the number of components.
           components   ;contains the list of components of this
                        ;prototype.
           cert_measure  ; the certainity with which the prototype
                         ;is confirmed.
           deg_of_belief   ;the certainity measure obtained in a
                           ;particular case.
           frame_considered_already_flag   ;flag used during the
                                           ;inferencing process.
   )
;-----------------------------------------------------------------
   (defstruct component  ;structure of a component being defined
           name_of_comp  ;associated name of the component.
           comp_val  ;value of the component in a particular case.
           type  ;type of the component.
           plausible_vals  ;the enumerated list of values, the
                           ;component can take.
           range_of_vals  ;the range in which the component value
                          ;must lie.
           default_val  ;default value of the component, if any.
           import_measure  ;importance measure of the component.
           percent_of_belief  ;the measure of belief about the
                              ;value of this component.
           percent_of_belief_present_flag  ;indicates the presence
                                           ;of %-of-belief for
                                           ;this component.
           val_rules  ;contains the list of value-fetching rules
                      ;for this component.
           trigger_rules  ;contains the list of trigger rules
                          ;which trigger other prototypes.
   )
;-----------------------------------------------------------------
```

Figure 4.1 Data Structures (in LISP)

```
;-------------------------------------------------------------------
   (defstruct rule  ;structure of the rule is being defined.
           name_of_rule  ;associated name of the rule.
           premise  ;antecedent part of the rule.
           action  ;action part of the rule.
           wt_factor  ;weightage-factor if any associated with the
                      ;rule.
           prompt  ;prompt to be presented to the user in case, it
                   ;is a `askuser' rule.
           percent_of_belief_update_flag     ;indicates   whether
                                             ;%_of_belief is being
                                             ;updated   in   this
                                             ;rule.
   )
;-------------------------------------------------------------------
```

Figure 4.1 Data Structures (in LISP)
(contd.)

4.1.2 Component

The second level of structure is defined as a component. Each component points to a subframe of knowledge at the object level. Component structure is also defined as a frame structure with its own fields. A brief explanation of the important fields in this structure are as follows:

* **Type**, which defines the type of the component. A component can be any of the four types viz. YES/NO, ENUMERATED, RANGE_VAR (range variable) and GENERAL.

* **Comp_val**, the value of the component which is deduced at the time of consultation using the value rules.

* **Plausible_vals**, which is defined for a component of type ENUMERATED. It contains the list of possible values that the component can take.

* **Range_of_vals**, which is defined for a component of type

RANGE_VAR. It is a list containing two elements, the first denotes the lower-limit and the second denotes the upper-limit respectively.

* **Default_val**, a slot which contains the default-value of the component if provided by the user. This is taken as the value of the component if an "unknown" response is obtained from the user.

* **Import_measure**, denotes the important measure i.e. the level of contribution of this component at the frame level. This is used in the confirmation of the prototype. This must be in the range of 0 to 10.

* **Percent_of_belief**, the %-of-belief about the value obtained for this component. If the value of the component is obtained by asking the user, this gives the system the measure of belief of that fact. It should be in the range of 0 to 100.

* **Val_rules**, which is one of the important slots. This slot contains the production rules which do their reasoning in a local production system environment to deduce a value for the component. The structure of these rules is described in the next section.

* **Trigger_rules**, which is the next important slot. This slot contains rules that trigger other prototypes with a particular degree of suggestivity. If this particular prototype is confirmed, then the prototypes triggered by these rules become `potentially relevant'.

4.1.3 Rules

These form the important slots of a component and hence worth mentioning. A brief description of the various fields in a rule structure and the kind of constructs and syntax that need to be followed in the framing of these rules is presented below.

The important fields in a typical value-rule are:

(a) **premise**, which holds the IF part of the rule.

(b) **action**, which holds the ACTION part of the rule.

(c) **wt_factor**, a weightage factor assigned to the rule used in conflict resolution. This is elaborated in the section on inferencing strategy.

The value-rules are of two kinds. These two kinds differ in their premise part. The first one consists of a set of conditions while the second consists of just `askuser' in the premise which denotes that the value should be obtained from the user. The syntax of premise is:

(<condition-1>...<condition-n>) or (askuser)

where each condition is any binary predicate. The form of the <condition-i> is as follows:

(predicate-i firstarg-i secarg-i)

The arguments of the binary predicate should be specified in a format illustrated below. If the argument is a component value,

then the expression format is

(frame-name -> component-name)

where `frame-name' is the name of the prototype which contains the particular component. If the argument is a slot value of the component, other than its value, then the format is

(frame-name -> component-name -> slot-name)

The action part can be any of the following two kinds of action statements:

(a) a simple action statement using the construct ASSIGN which performs a single action. The ASSIGN construct essentially assigns a value to a slot. &

(b) a compound action statement using the construct DOALL which enables to perform multiple actions in a single statement.

The ASSIGN construct can have one or two arguments. The ASSIGN construct is used with a single argument if the premise is of `askuser' type. The syntax for a simple action statement is:

(ASSIGN <arg-1>) or (ASSIGN <arg-1> <arg-2>)

where <arg-1> and <arg-2> follow the same syntax described above for the arguments.

The syntax for a compound action statement is:

(DOALL <simple-stat-1> ... <simple-stat-n>)

where <simple-stat-1> to <simple-stat-n> are simple action statements.

The trigger rules are of a single kind with a simple syntax. The premise is simply a set of conditions i.e. same as the premise part of the first kind of value rules. The action part can again be of two kinds of statements. (a) The simple action statement this time uses the TRIGGER construct. The TRIGGER construct essentially triggers a prototype with a certain degree of suggestivity. The degree of suggestivity is the measure of likeliness that the triggered prototype is worth investigating. The degree of suggestivity must lie in the range of 0 to 100.

The syntax of the simple action statement for a trigger rule is:

(TRIGGER prototype-name degree-of-suggestivity)

(b) the compound action statement is the same as in the case of the value rules except that the simple statements present in it must be the simple action statements defined for the trigger rules above.

For examples, please refer to the snapshots presented in the users manual.

4.2 Knowledge Update Facility

The Knowledge Update Facility essentially is made up of a `Creating a Knowledge Base Module' and a `Edit a Knowledge Base Module' which are described below in detail.

The other features include a DELETE-KB function which deletes a selected knowledge base. In addition a graphical view of the selected knowledge base with scrolling facilities is also available. A snapshot of the graphical view is included in the User's Manual for reference.

4.2.1 Creating a New Knowledge Base Module

This module can be invoked from the main menu itself. When this is selected, the user is intending to create a new knowledge base. Firstly, this module prompts the user for the domain-name i.e. a name for the new knowledge base being created, followed by the root_frame name and the link frames of the root_frame. An entry is made for the new domain name in the list of current knowledge bases. Prototype structures are created for the root_frame and its link frames with the necessary initializations. Files are created to hold the global-data for this new knowledge base and also for the root_frame and its link frames. Then the user is advised to go ahead with the updating (i.e. editing) of the new knowledge base created by selecting it on the main menu.

4.2.2 Edit a Knowledge Base

This module enables the knowledge engineer to perform a variety of editing operations like reading, adding, deleting and modifying on the knowledge base. In order to perform the editing operations on a particular knowledge base, it should be first selected at the main menu level. Then selecting the EDIT-KB option on the second level menu invokes this module. All the editing operations are provided at three different levels i.e. at frame level, component level and rule level respectively. The detailed explanation of the various editing operations at different levels is provided in the User Manual with the aid of snapshots.

4.3 Inferencing Strategy

To begin with, the root_frame prototype which is the root of the prototype network is selected as the current prototype. In this section, the words "prototype" and "frame" are being used interchangeably. The root_frame contains mostly the initial data. One important thing that must be noted, is that the inference strategy followed for the root_frame is slightly different from the strategy that is followed for the remaining frames. This strategy difference lies in the way in which the components are handled. First, the two strategies followed are presented, followed by a discussion on the reasons for the difference in strategy.

Before the two strategies are presented, the two important

support modules viz. "Fill Component" module and the "Confirm Component" module are discussed.

4.3.1 Fill Component Module

The main function of this module is to fill the component i.e. deduce a value for the component, which is passed to it as an input parameter using the value rules present in the val_rules slot of the particular component. The various steps involved in the filling process are described in the following algorithm.

Algorithm fill-component(current_component)

Step 1: Get the value rules associated with the current_component.
Step 2: From the set of rules, determine the Conflict set.
Step 3: Using Conflict resolution strategy, select the rule to be fired in this case. The remaining ones are stored in one of the slots of the component so that they can be retrieved during backtracking.
Step 4: Check the type of the value rule.
Step 5: If it is `askuser' type, prompt the user for the necessary information and fill the value of the component
else
apply the rule i.e. execute the action part thereby deducing a value for the component.

In the above described filling process, the conflict set is formed by accumulating the rules whose premise part is satisfied. When there are more than one member in the Conflict set, the Conflict resolution strategy is applied. The resolution strategy involves the following steps:

1. The conflict set is ordered by the weightage factors associated with the rules, if any. The default value of the

weightage factor is taken as 0.

2. If the conflict is still not resolved, then the ordering is done based on the number of conditions present in the premise part of the rules.

3. Lastly, an arbitration is done.

4.3.2 Confirm Component Module

This module essentially performs the necessary checking on the value, of the component that is sent as the input parameter to it. The main aim of this module is to judge whether the value of the particular component is consistent or not. Different types of checking is done depending on the type of the component which include checking of the value to be within the range limits for a component of type RANGE_VAR, checking the value to be one of the plausible values for a component of type ENUMERATED. If the value is found to be consistent, then the component is marked as confirmed. else it is marked as not confirmed. Similarly, if a "unknown" response is present as the value of the component, a default-value if present is assigned as the value of the component and the component is marked as confirmed else it is marked as not confirmed.

4.3.3 Inferencing at Root-Level

In this sub-section, the inferencing strategy followed at the root level is described. As mentioned earlier, the root_frame is made the current prototype to begin with and the following

algorithm is applied to carry on the inferencing process.

Algorithm inferencing_at_root_level

Step 1: Get the Order of Filling of the components of the root_frame.
Step 2: While still there is a component unfilled, do
else
go to step 7.
Step 3: Fill the current component.
Step 4: Confirm the current component.
Step 5: If the component is not confirmed, go to Step 6.
5.a: Fire the possible trigger rules to check if any one of the prototypes are triggered.
5.b: If none of the prototypes are triggered, go to Step 6.
5.c: From the triggered prototypes, separate the link level prototypes and check for deeper level prototypes.
5.d: If none of them triggered, add the link level triggered prototypes to the hypothesis list and go to Step 6.
5.e: Order the deeper level triggered prototypes based on the degree of suggestivity.
5.f: While there is a deeper level triggered prototype, do
5.g: Switch the current frame to the most suggested deeper level triggered prototype and call the inferencing strategy at hierarchial level with this frame as the argument.
Step 6: Go to step 2.
Step 7: The root_level inferencing is complete. If any link level frames are triggered, order them based on their degree of suggestivity and call the inferencing at hierarchial level with the prototype that is present on the top of the hypothesis list.

4.3.4 Inferencing at hierarchial level

In this sub-section, the second type of the inferencing strategy which is followed for the rest of the prototypes other than the root_frame is presented. The various steps involved in this inferencing strategy is explained in the algorithm given below. The algorithm is followed by the next sub-section in which the reasons for following two different strategies is explained.

Algorithm inferencing-at-hierarchial-level (current_frame)

Step 1: Get the order in which the components of the current_frame are to be filled up.
Step 2: While still there is a component unfilled,
2.a: Fill the current component.
2.b: Confirm the current component.
Step 3: The degree of belief of the current prototype is computed by collapsing the sum of the product of the importance measures and the percent-of-beliefs of all the components of the current prototype.
Step 4: If the degree of belief is greater than or equal to the threshold measure of the current prototype, then the degree of belief is set to the certainity measure of the current prototype and the prototype is marked as confirmed
else
{
the current prototype is marked as not confirmed and Go to Step 6.
}
Step 5: Here, the prototype is confirmed.
5.a: If there are no link frames to the current prototype, then it can be concluded that the system has arrived at a solution. Then the list of confirmed prototypes along with their degrees of belief and action information are printed as conclusions and the user is asked whether he wants to have another solution?.
else
{
Fire the possible trigger rules of all the components of the current prototype, to check if any of the prototypes have triggered. If yes, order them according to their degrees of suggestivity and add them to the top of hypothesis list. Then call the same algorithm making the prototype on the top of the hypothesis list as the current prototype.
}
5.b: If another solution is sought, then the system backtracks along the path just concluded to see if any alternate solution is possible. If none is found, it continues with the remaining prototypes left on the hypothesis list.
else
{
End the inferencing process.
}
Step 6: Here, the current prototype is not confirmed.
If the hypothesis list is empty, then
End the inferencing process.
else
{

```
Check if the current prototype is invoked from the
root_frame.     If     YES,     call     the     algorithm
inferencing_at_root_level.
 }
                    else
              {
Call  the same algorithm making the prototype on
the  top of the  hypothesis  list as the current
prototype.
 }
```

4.3.5 Reasons for the Adoption of Two Strategies

The reasons for following a different inferencing strategy at the root level are listed below:

* The root_frame consists mostly of initial data.

* There are no concepts of certainity measure and degree of belief defined for this prototype.

* Each component defined in this prototype plays a vital role in the inferencing process. Hence they are handled in a special way.

* The switching to a deeper level prototype is being done here. The reason is that since strong evidence is being provided, an early solution may be possible. One can argue that why can't the same strategy be adopted at the hierarchy level. The argument is genuine but due to the implementation overhead, it is being limited only to the root level. Another advantage obtained by limiting it to the root level is that the context switching becomes simpler and the overhead is reduced.

For example, Assume the GENERAL SYMPTOMS prototype described in the last chapter as the root_frame and that its components viz. BLOOD PRESSURE and BODY TEMPERATURE are filled. Suppose a very

high value of BLOOD PRESSURE is obtained in a particular case and a trigger rule had suggested HEART DEFECT which is a deeper level frame in the hierarchy. Then according to the strategy adopted in this shell, the context switches to the prototype HEART DEFECT making it as the current prototype.

4.4 Explanation Facility

The questions posed to the system by the user includes the HOW and WHY.

The WHY question is answered with the aid of a WHY-STACK. As mentioned earlier, Interpretation of WHY question is of two kinds depending on the context and the review stage. When the question is posed in the context of a particular prototype, i.e. resulting in an interpretation of "Why are you asking for this information?", it is answered by displaying the status of the current frame i.e. answers by saying that it is trying to confirm the current frame. It also displays the components that are filled till then and the components that are yet to be filled. In the review stage i.e. when the interpretation made is "Why are you considering this prototype?",the reason is generated using the WHY-STACK which is explained below.

The WHY-STACK always contains the latest confirmed prototypes list. Along with each prototype, information about its source of triggering is stored. The updating of the WHY-STACK is done in parallel with the hypothesis list. Each entry in the WHY-STACK is a list of five elements. They include the prototype name,

degree of suggestivity, trigger rule responsible, the component and prototype names which contain this trigger rule. When the question is posed, the related entry is retrieved from the WHY-STACK and the explanation is presented.

Explanation for the HOW question is answered with the aid of a HOW-STACK. The HOW-STACK always keeps track of the list of components of the current prototype which are filled. Each entry in the HOW-STACK is a list of two elements, the first is the component name and the second, the value-rule which deduced the particular component's value. This option is available to the user only after the first WHY question is posed. Since it is interpreted as "How the value of a particular component is deduced?", if this option is selected then the list of components which are filled is displayed and the user is asked to enter the component name to be explained. From the HOW-STACK, the entry of the component to be explained is retrieved and the corresponding rule is displayed to the user thereby completing the HOW explanation.

The explanation is provided in a separate window during the consult mode. The graphical view of the knowledge base is also displayed in another window with the current frame boxed. Thus the interactive view of the knowledge base is provided during the inferencing process with appropriate explanation facility. The snapshot showing the interactive view is included in the User's Manual for reference.

4.5 HELP Feature

Help feature is available at all levels. In some of the menus, a help window is present which provides help in explaining the functions of the various options available at that level and in the rest, it is implicit in nature i.e. the options themselves explain their functions.

CHAPTER 5

CONCLUSIONS

5.1 Conclusions

In the expert system shell architecture implemented, emphasis was laid in finding a suitable knowledge representation scheme for a domain application like medical diagnosis. The pros and cons of different knowledge representation schemes were observed. The advantages of using more than one knowledge representation scheme were also noted. Already, a lot of work has been done in the past decade on the same issue and many ideas have been put forth.

The knowledge representation scheme used is a combination of frame and rules into a single data structure called the prototype. Many medical texts present diseases by discussing typical cases. It is relatively easy to express knowledge about diseases in terms of prototypes. Different types of knowledge can be separated in this formalism. The whole idea is that the prototypes should be able capable of being used to capture knowledge about a very wide class of situations.

The implementation of the expert system shell gave a good opportunity to look into the various issues that must be taken care of, in the shell organization. One of the aims of the proposed shell architecture was that it should be able to tune the architecture for different domains by slight modifications. Though it has not been fully realized, it has been observed that the

domain effect i.e. the influence of the domain for which the shell is being designed is very high. In other words, if a general purpose shell is to be implemented, the issues to be considered are complicated.

The level of user interface that should be incorporated in the shell is another important issue that has been considered. Facilities should be such that the knowledge bases can be created and updated easily. Also during the consultation mode, an interactive view of the knowledge bases must be provided so that the user can get a view of the control flow of the inferencing process. In the work done, though it has been implemented on a workstation, the advantage gained is only multiple windows. This is due to the poor quality of the graphic facilities for text processing. It has been observed that the graphic facilities needed for shell development must be more suitable for text processing.

The suitable inferencing strategy that has to be adopted for a representation scheme like prototype is "hypothesize and match". A explanation facility which answers the questions HOW and WHY is incorporated. The complexities that lie in providing a good explanation facility has been noted.

5.2 Extensions & Scope for Further Work

In this thesis work, the basic frame-work of the proposed expert system shell architecture along with some features of user interface has been developed. Some of the ways in which

improvements can be made on the shell are discussed below.

The first and the most important work that could be done using this shell is to experiment with the architecture by developing an expert system for medical diagnosis or fault diagnosis. This could bring out more issues that one has to look into, if the shell is to be further tuned to this application. There are various measures defined in the shell like the importance measure, certainity measure, degree of suggestivity etc. The evaluation of these measures can only be done by proper experimentation and concrete results can be established. Aspects of learning these parameters through examples and validation may also be explored.

The incorporation of scripts in the shell architecture is another direction that should be explored. Reasoning based on a script would represent the causal situation. These may be used to develop expectations about the data and also for generating explanations. Causal knowledge plays a very important role in diagnosis. We may explore ways and means to incorporate causal network to provide context switching for frames and for generating explanations.

The user interface part also needs improvement. Facilities which enable the user to interact with the knowledge base graphically need to be added. An important facility that could be added is the "trace facility" during the inferencing process. This provides the user, a facility to know in more detail about the inferencing process. This can be also useful in providing the

debugging assistance to the knowledge engineer.

A final comment is on the explanation facility. Options like WHAT or CLARIFY which will present information about the knowledge base with a good natural language interface can be incorporated.

REFERENCES

1. Aikins. J.,
"Prototypical Knowledge for Expert Systems",
Artificial Intelligence , 20, 163 - 210. [Aikins83]

2. Beach. S.,
"A Comparision of Large Expert System Building Tools",
Sprang-Robinson Report-2, 2(10) (October 1986). [Beach86]

3. Branchman. R ., and J.Schmolz,
"An Overview of KL-ONE Knowledge Representation",
System Cognitive Science. April 1985a. [Branchman85a]

4. Brownston. L ., R . Favell, E . Kant and N . Martin,
"Programming Expert Systems in OPS5",
Addison-Wesley, Reading, Mass., 1985. [Brownston85]

5. Clocksin. W., and C.Mellish,
"Programming in Prolog",
Springer-Verlag, NewYork 1981. [Clocksin84]

6. David W. Rolston
"Principles of Artificial Intelligence and Expert Systems Devlopment",
McGraw-Hill Book Co., 1988. [Rolston87]

7. Edward. H. Shortliffe and Bruce G. Buchanan,
"Rule-based Expert Systems",
Addision-Wesley Pub. Co., 1985. [Shortliffe85]

8. Furukawa. K., A. Takenchi, S.Kunifuji et. al,
"MANDALA: A Logic Based Knowledge Programming System",
Proceedings of the International Conference on Fifth Generation Computer Systems, 1984. [Funikawa84]

9. Karna. A.,
"Evaluating Existing Tools for Developing Expert systems in PC Environment",
Proceedings of the Conference on Expert Systems in Government,
IEEE Press, 1985. [Karna85]

10. Michie. D.,
"Introductory Readings in Expert Systems",
Gorolon and Breach, NewYork, 1984. [Michie]

11. Patel-Schneider. P.,
"Small can be Beautiful in Knowledge Representation"
Proceedings of the Workshop on Principles of Knowledge Based Systems", IEEE Press, 1984. [Patel84]

12. Pau. L,
"Survey of Expert Systems for Fault Detection, Test Generation and Maintenance",
Expert Systems, 3(2): 100-111 (April, 1986). [Pan86]

13. Peter Jackson
"Introduction to Expert Systems",
Addision-Wesley Pub. Co., Inc 1986. [Jackson85]

14. Pople. H,
"On the Mechanization of Abductive Logic",
IJCAI82, 147-152, 1982. [Pople82]

15. Robinson. J. A,
"A machine-oriented logic based on the Resolution Principle"
Journal of ACM, 12, 23-41. [Robinson]

16. Waterman. D,
"A Guide to Expert Systems"
Addison-Wesley, Reading, Mass., 1986. [Waterman84]

APPENDIX A

USER'S MANUAL

This appendix familiarizes the user with the various features that are available in the shell.

To begin with, the process of invoking the shell is explained. You have to first enter the lisp environment in order to invoke the shell. This can be done by typing lisp at the command line in the C-Shell. After entering the lisp environment, type

(load "shellfiles")

at the lisp prompt. When the prompt appears again, type (expss) to invoke the shell.

A.1 First level Menu

Once the shell is entered, the main-menu appears, a snapshot of which is shown in the Figure A.1. The various options available at this level are displayed at the top of the menu. Also the current knowledge bases which are available in the shell are listed near the middle of the screen followed by the option for creating a new knowledge base.

In order to know, how to select the options, press '3' which is the HELP option. The other alternative is use the mouse and take the cursor into the HELP box. Then press the leftmost button of the mouse to select this option. A HELP WINDOW is

created at the bottom of the screen which explains, how to select the options in general and know about them.

A brief description of the various options available at this menu is given below.

[DELETE-KB] Deletes the selected knowledge base.

[SAVE-KB] Saves the current knowledge base in memory.

[HELP] Explains how to use the menu.

[EXIT] To exit from the shell.

In order to select a particular knowledge base, take the cursor into the box containing the knowledge base name and press the leftmost button of the mouse. Similarly, if you want to create a new knowledge base, select the option 'ADD-A-NEW-KB'.

A.2 Second level Menu

If you select a particular knowledge base, then you enter the second level menu which is as shown in the Figure A.2. Description of the options available at this level of menu is given below.

A.2.1 EDIT-KB

This option enables you to perform the editing operations which include adding, modifying, reading and deleting operations, on the selected knowledge base. All the editing operations are provided at three different levels i.e. at the frame level, at the component level and at the rule level. When this option is

selected, the editing options at the frame level are displayed in a separate window which appear as:

1. ADD / CHANGE A FRAME.
2. DELETE A FRAME.
3. READ A FRAME.

A snapshot of READ A FRAME in which a selected frame with all the important fields along with their values, listed in a neat format, is shown in the Figure A.3. DELETE A FRAME deletes a selected frame. If the option ADD / CHANGE A FRAME is selected, then another menu which provides options for adding or modifying the various slots of a selected frame, a snapshot of which is shown in Figure A.4, is displayed.

The EDIT A COMPONENT option as shown in Figure A.4 if selected, takes you to the next level i.e. to the component level. The various editing operations available at the component level are same as those at the frame level i.e.

1. ADD / CHANGE A COMPONENT.
2. DELETE A COMPONENT.
3. READ A COMPONENT.

A snapshot of READ A COMPONENT in which again all the important fields are listed is shown in the Figure A.5. If the option ADD / CHANGE A COMPONENT is selected, then again another menu which provides options for adding or modifying the various fields in a selected component, a snapshot of which is shown in Figure A.6, is displayed to the user. DELETE A COMPONENT deletes a selected component.

The options 'EDIT A VALUE-RULE' AND 'EDIT A TRIGGER-RULE' available at the menu shown in the Figure A.6, take you to the third and final level of editing i.e. to the rule level. Here, again all the editing operations are available at the user's disposal. The snapshots of READ A VALUE-RULE and TRIGGER-RULE which display a selected value-rule and a selected trigger-rule respectively are shown in FIgures A.7 & A.8.

Thus the organized manner of editing facilities makes the user's work easy in editing the selected knowledge base.

A.2.2 DISPLAY

This option if selected displays to you, the graphical view of the selected knowledge base. Scrolling feature available, enables you to move the graph and unveil the hidden portions of the graph, if any, which are not covered in the current display. Hints for making use of the scrolling feature are displayed in a box adjacent to the graphical view. A snapshot of this graphical view is shown in the Figure A.9.

A.2.3 CONSULT

This option if selected, runs a consultation with the selected knowledge base. During the consultation, explanation is provided by the display of the interactive view of the knowledge bases. A graphical view, in which the current frame that is being considered is highlighted (i.e. boxed), is displayed in one window. The information that is required from the user is prompted

appropriately, in the **CONSULTATION MENU.** When options like WHY and HOW are selected, explanation is provided in a separate **EXPLANATION WINDOW.** A snapshot of the shell during the consultation mode is shown in the Figure A.10.

A.2.4 HELP

This is the same HELP option that is available at the first level menu.

A.2.5 TO TOP MENU

This option if selected, takes you to the first level menu i.e. enables you to select the options available at the first level menu.

--*---*--

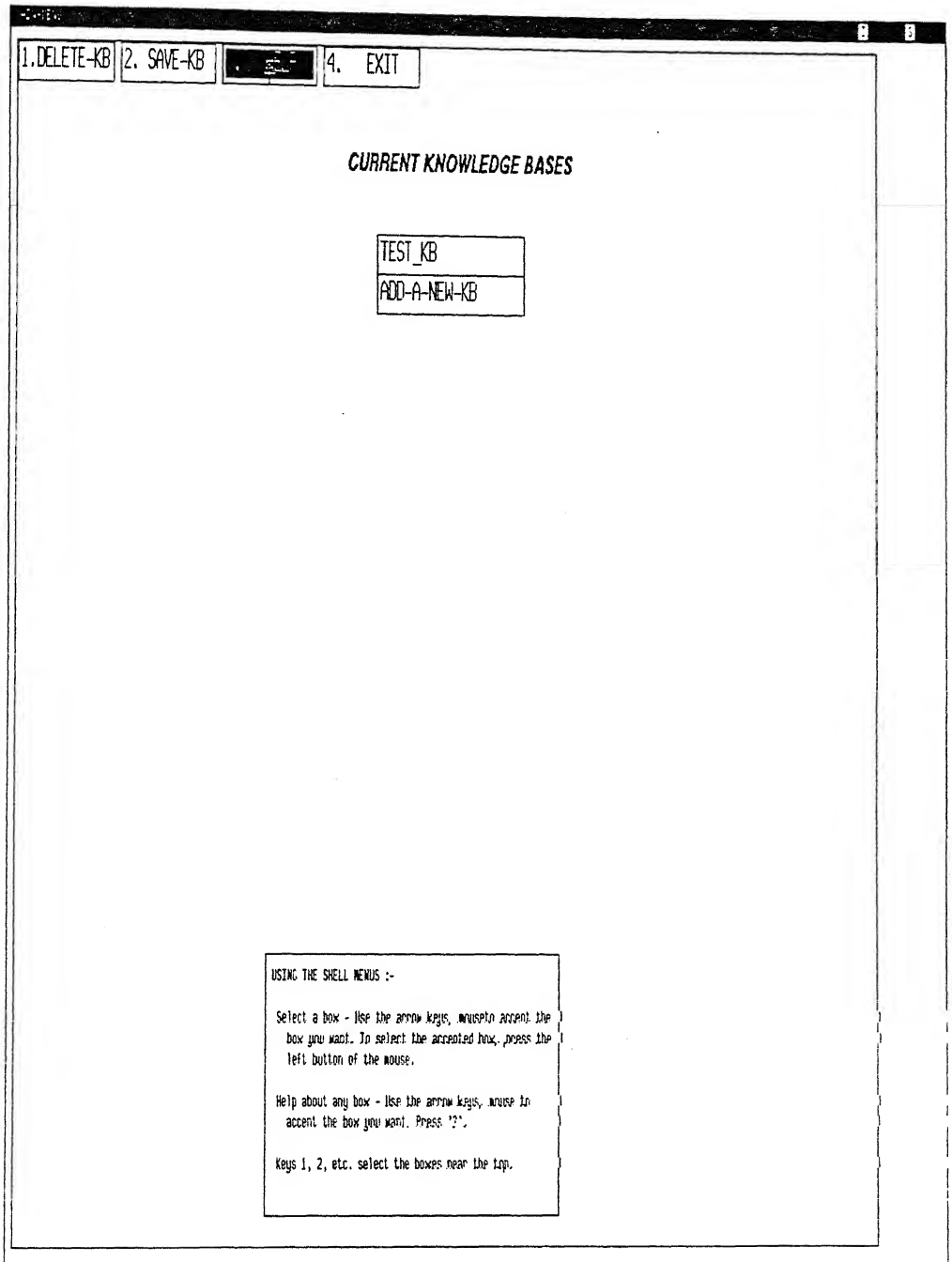


Figure A.1 The First Level Menu.

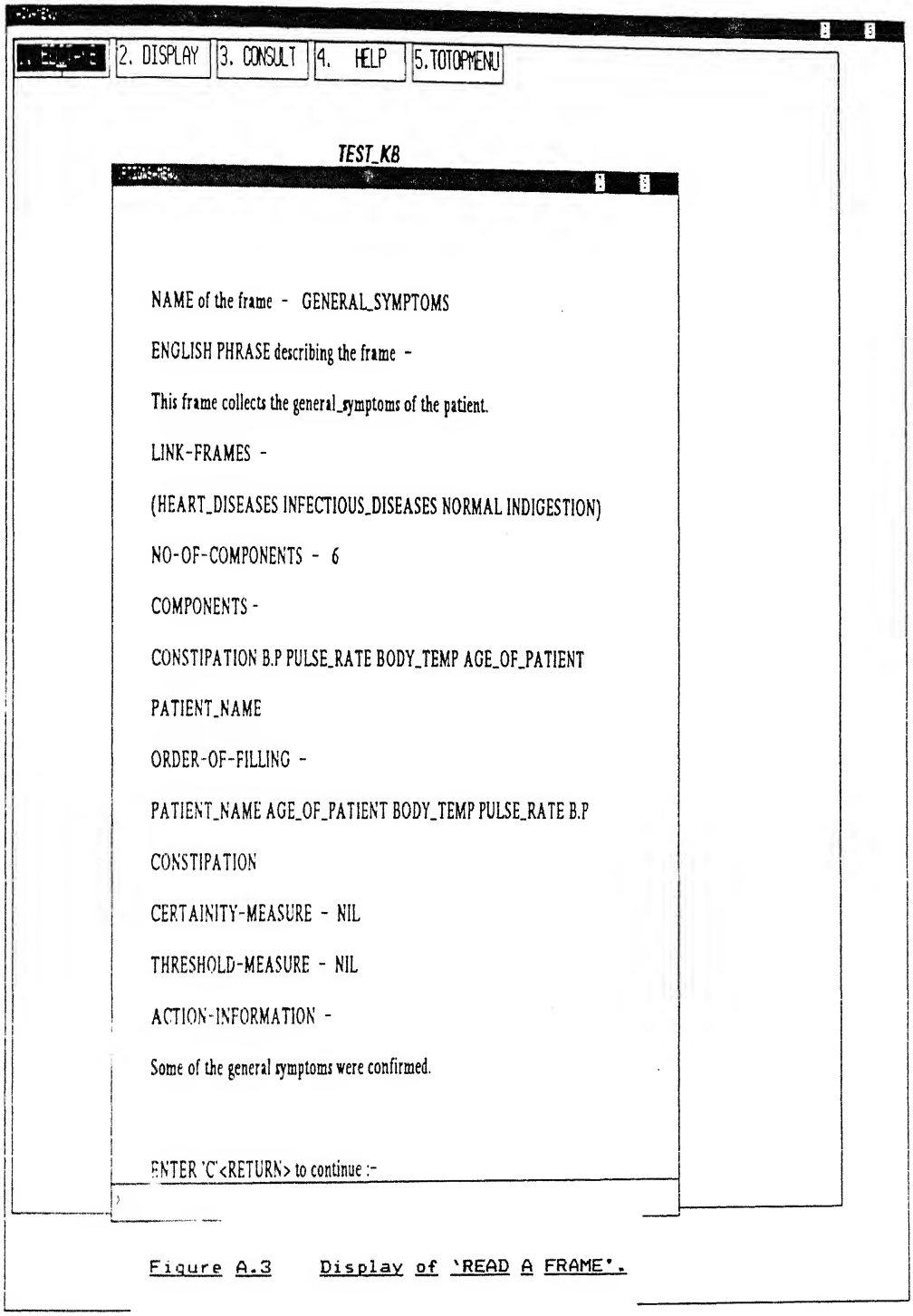


Figure A.3 Display of 'READ A FRAME'.

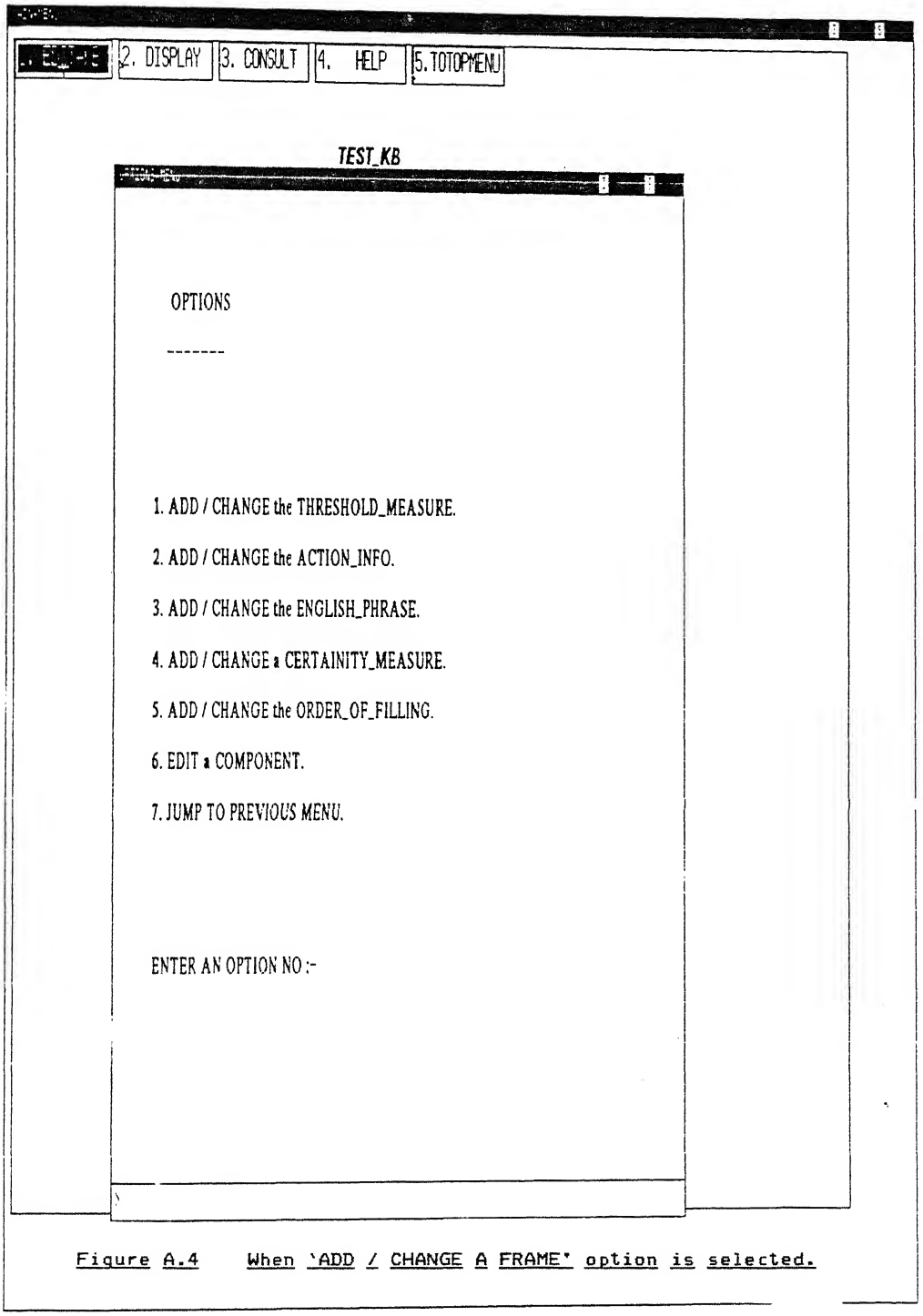


Figure A.4 When 'ADD / CHANGE A FRAME' option is selected.

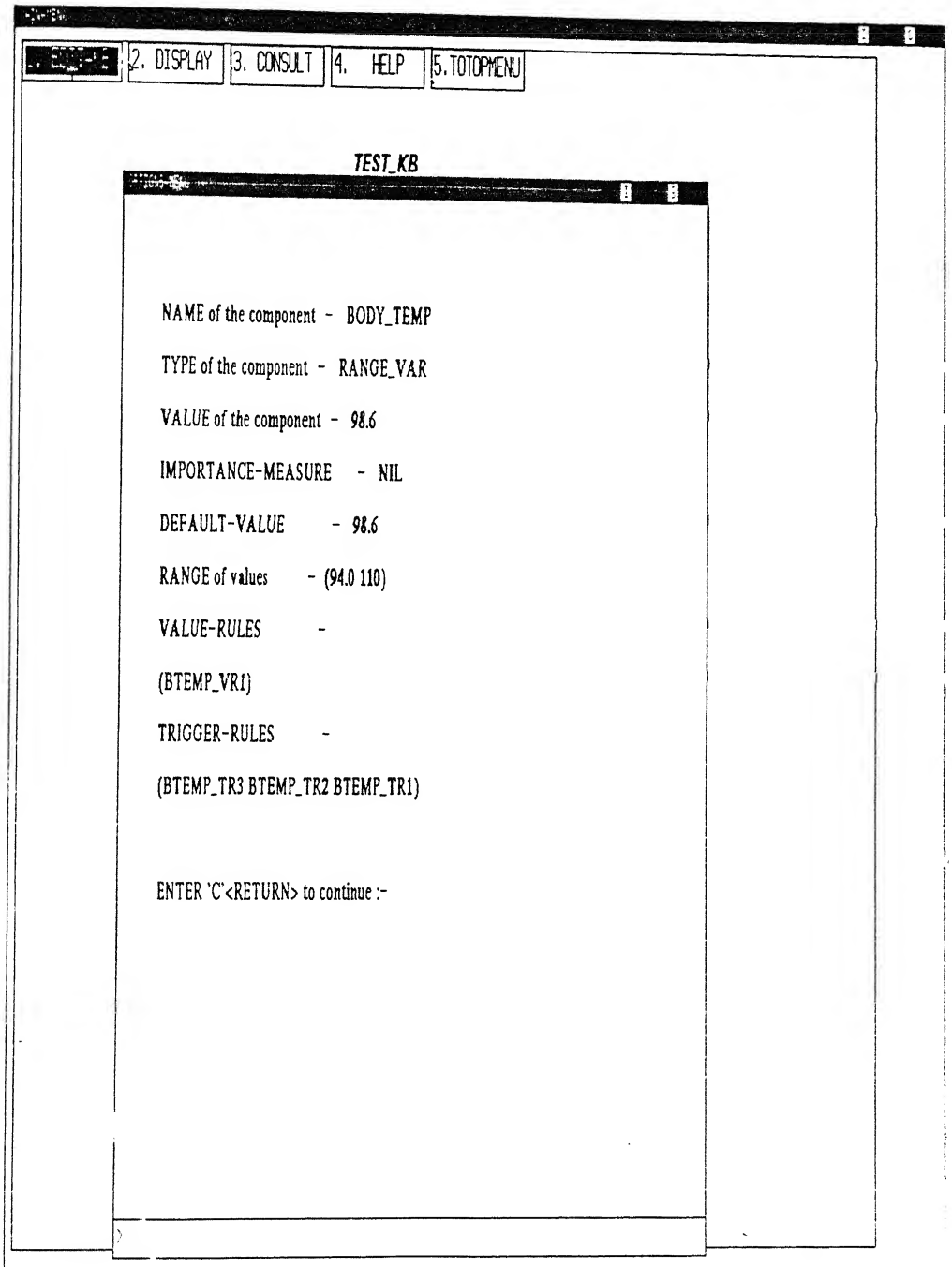


Figure A.5 Display of 'READ A COMPONENT'.

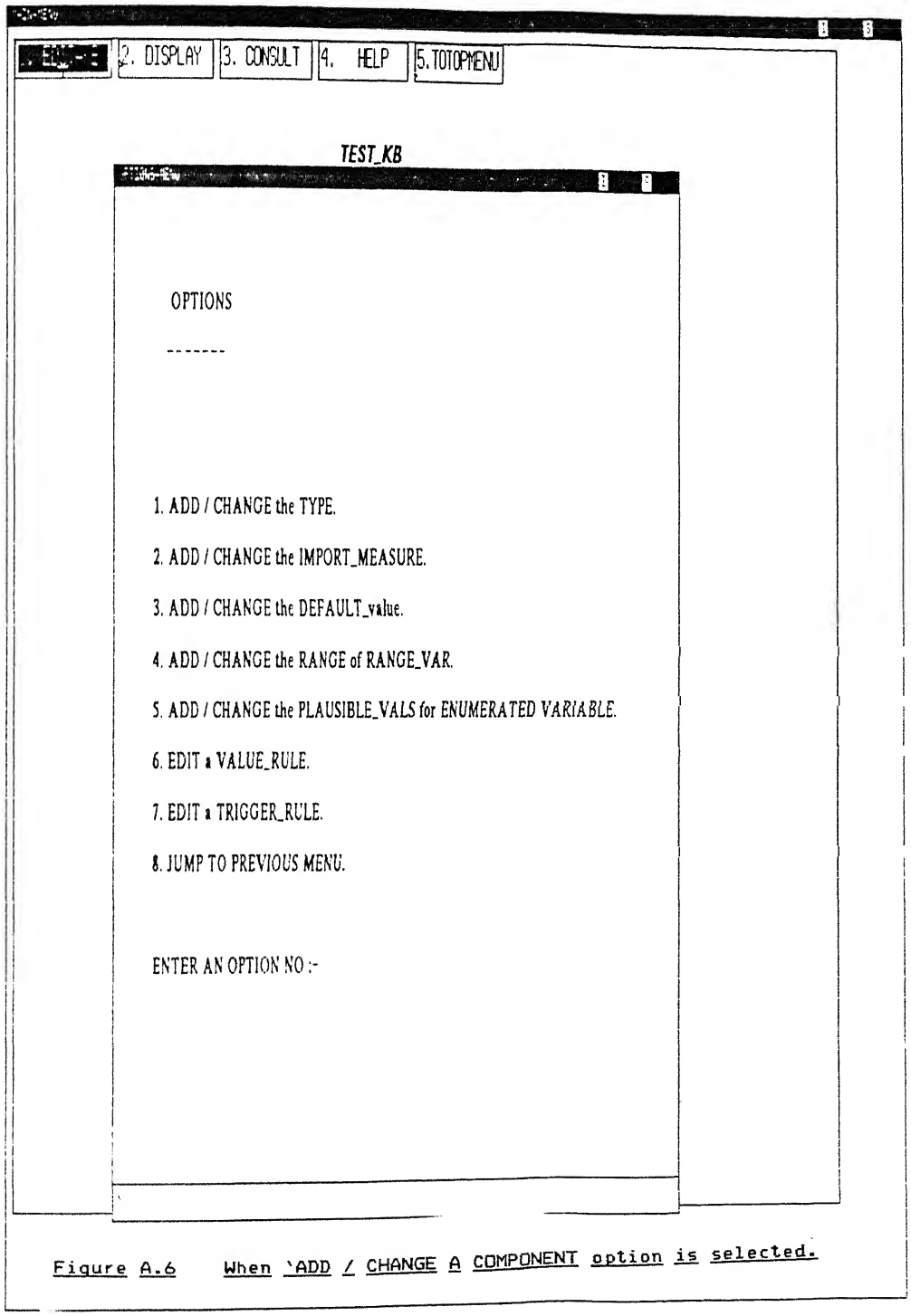


Figure A.6 When `ADD / CHANGE A COMPONENT option is selected.

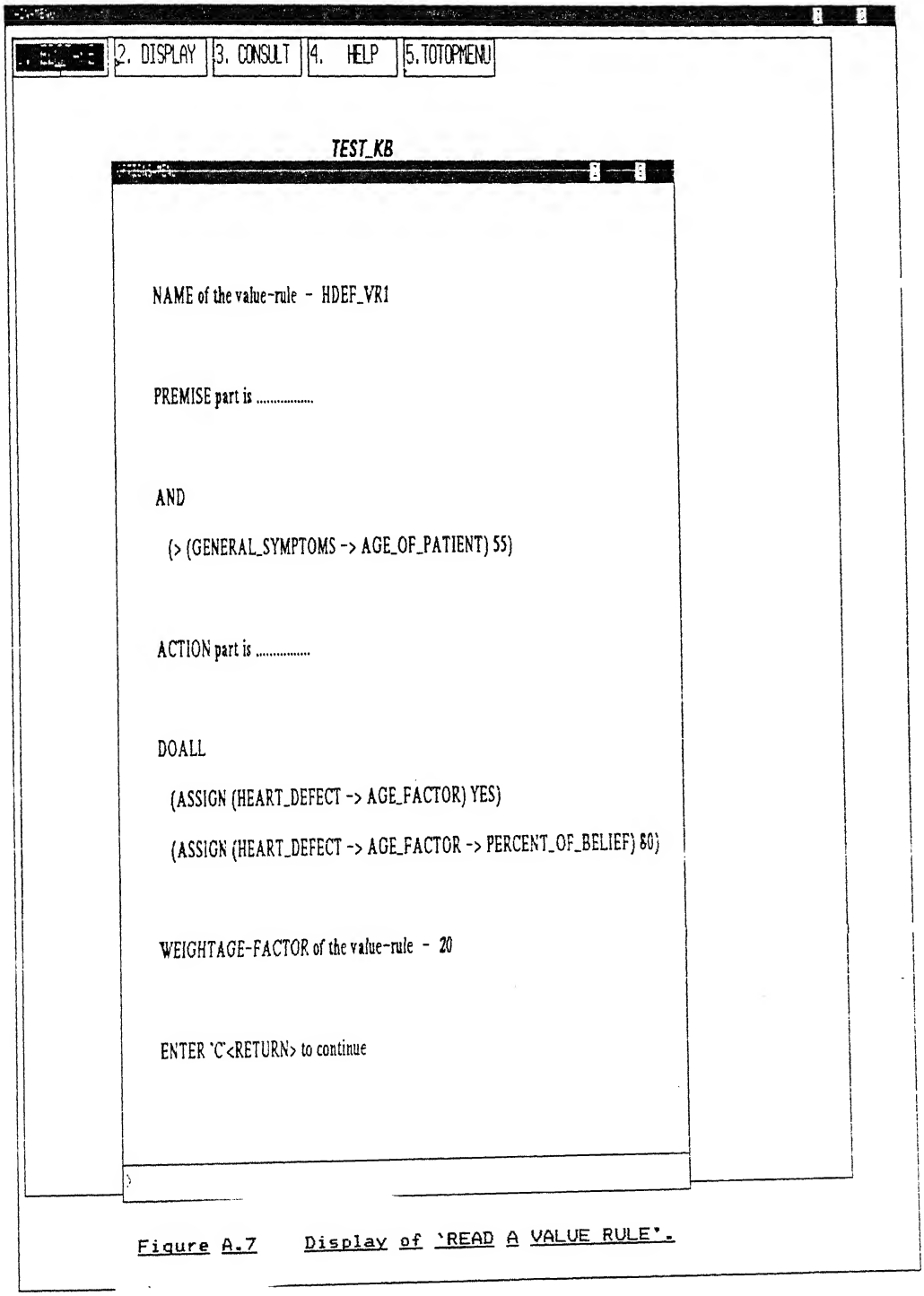


Figure A.7 Display of 'READ A VALUE RULE'.

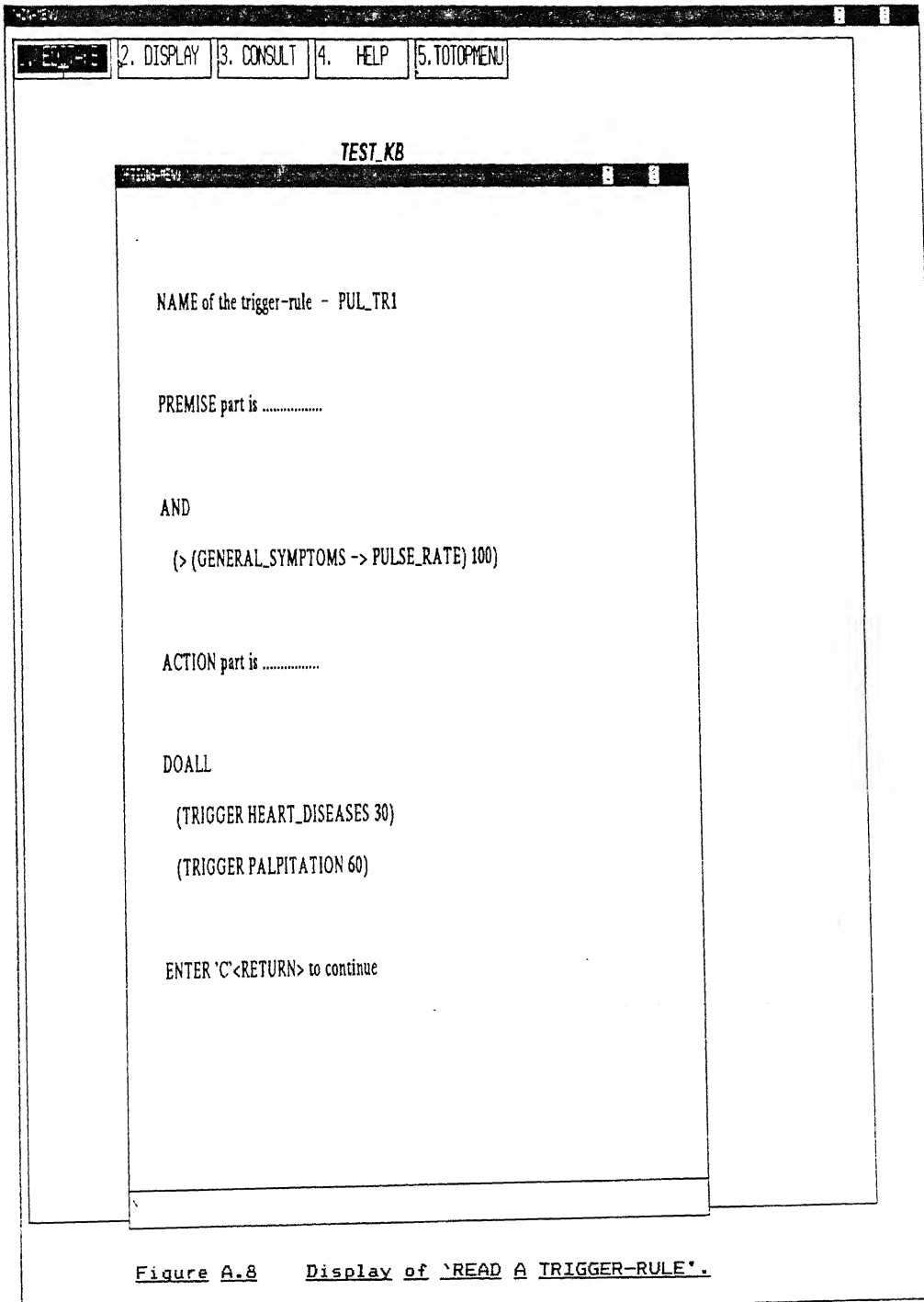


Figure A.8 Display of `READ A TRIGGER-RULE'.

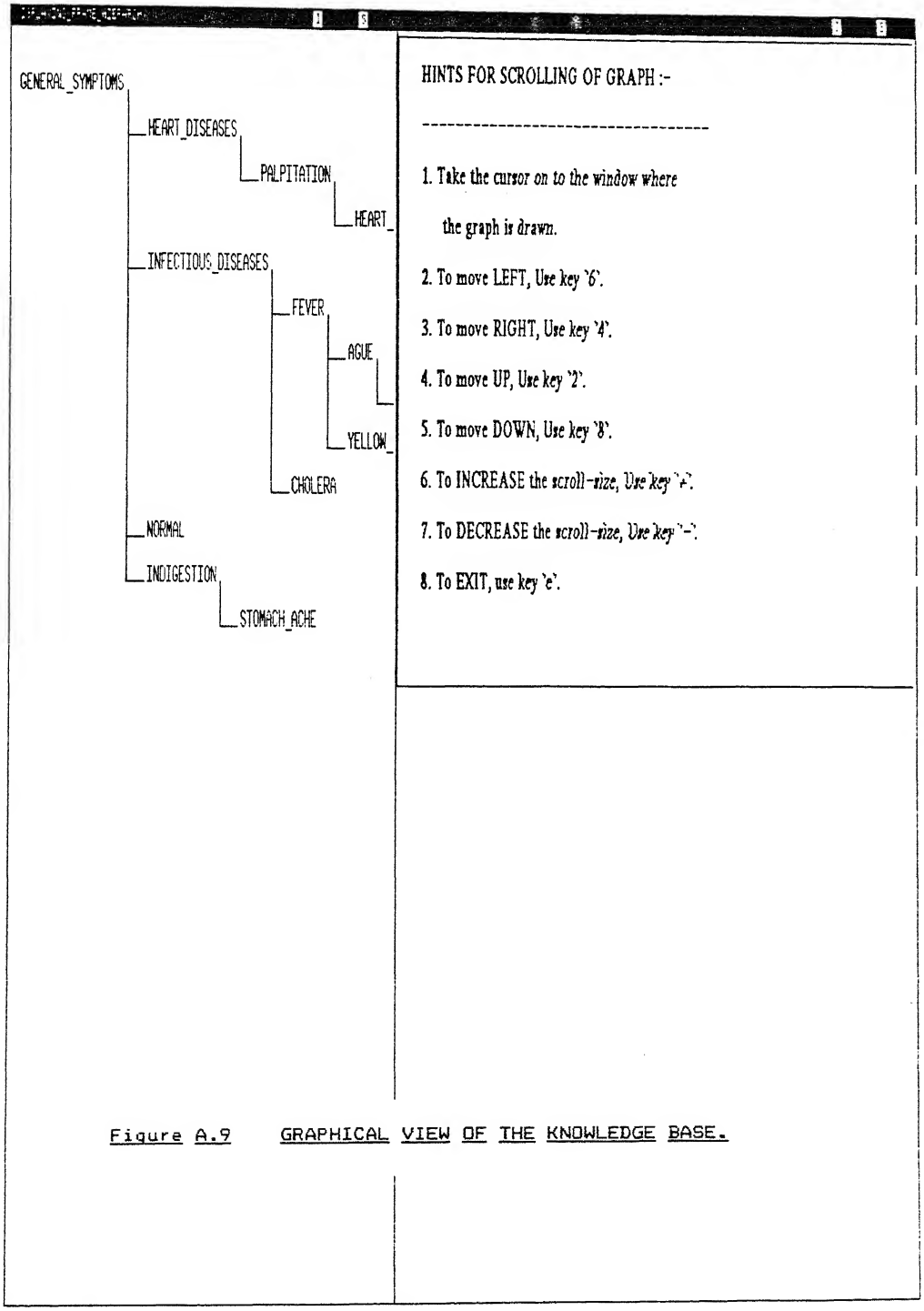


Figure A.9 GRAPHICAL VIEW OF THE KNOWLEDGE BASE.

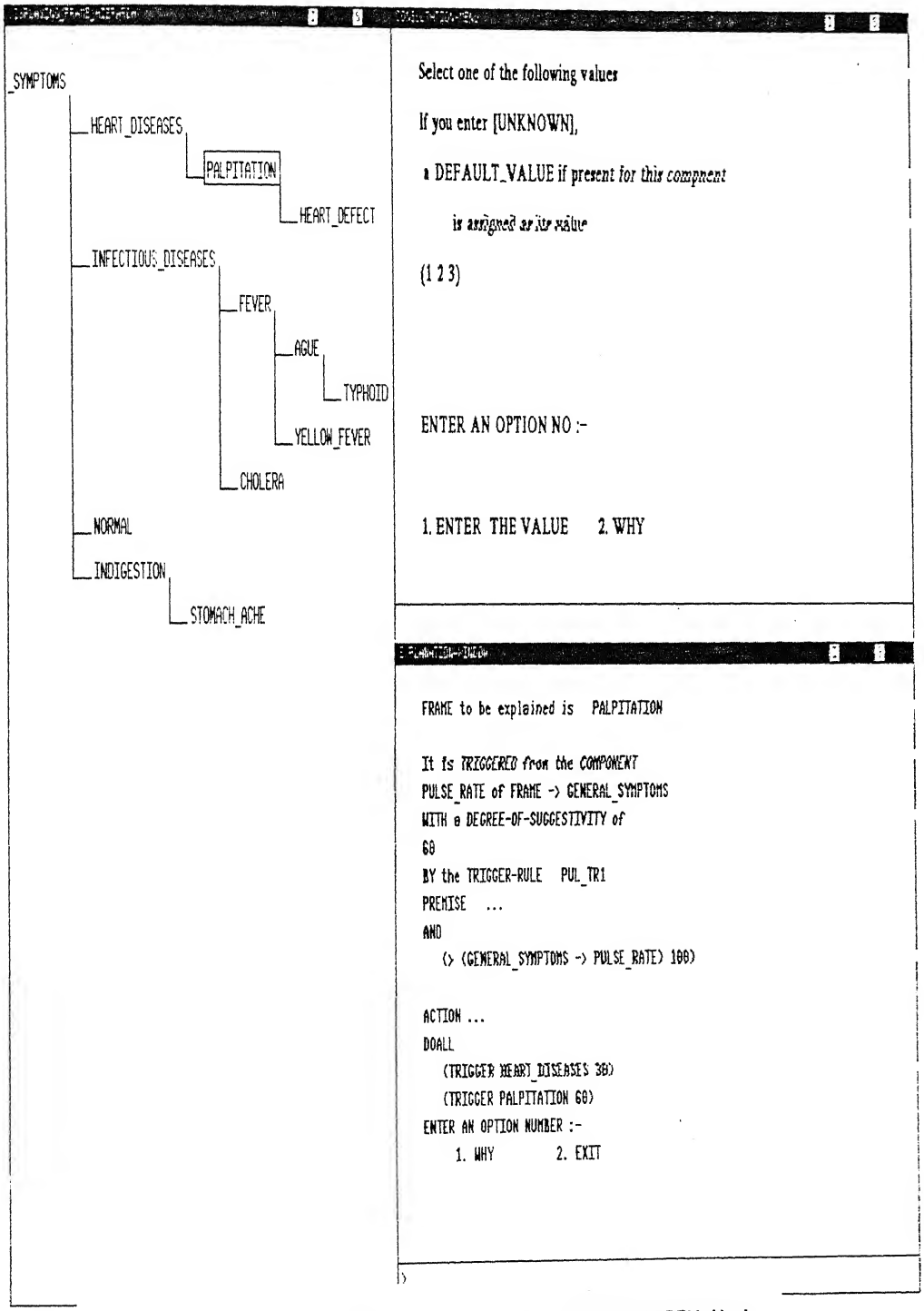


Figure A.10 EXPLANATION during the CONSULTATION Mode.